Unit-3.

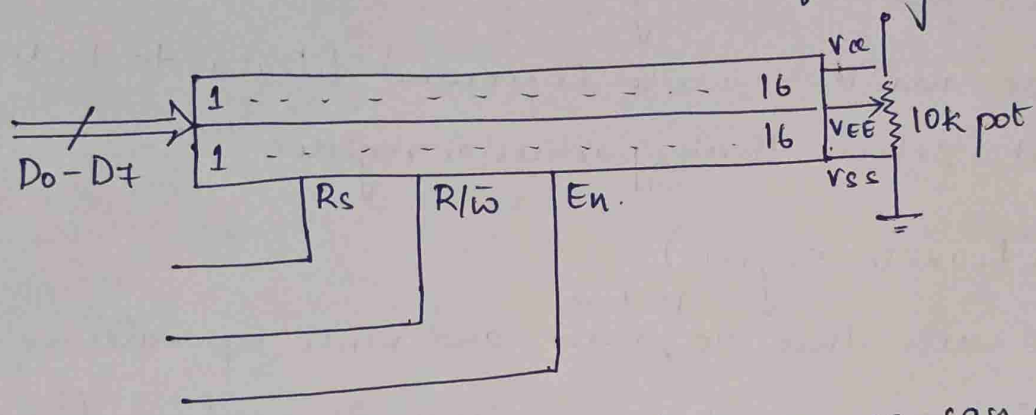
8051 interfacing and Application

* Basics of LCD interfacing :-

- In recent years the LCD is used widely as it replaces LED's (7 segment LED display) this is due to the following reasons.

1. declining prices of LCD.
2. Ability to display number, character, graphics
3. Ease of programming for character and graphics.

Here we consider a 16 x2 LCD interfacing :-



- It has two rows and in each row we can display 16 char.
- It has 8 bit data bus (D0-D7)
- It has 3 control signal Rs, R/W, En. (Register Select, Read/Write, Enable)
- To change the brightness of LCD, 10KΩ potentiometer is connected to Vcc.

  pot has 3 terminals to connect

  1. Connected to Vcc , 2. Variable terminal of pot Connected to VEE
  3. Gnd terminal is connected to Vss

- By varing the resistance of the pot we can vary the brightness of the LCD

* LCD Control Signals.

- LCD has two internal resistance

1. Command resistor – user can store the command such as clear display, display on cursor on, Select LCD 16X2

2. Data register – user has to store the ASCII value of character that we wana display on LCD

- To Select either of the two register Rs control signal is used. Rs → Register Select terminal.

Rs = 0 – Command reg is selected

Rs = 1 – Data register is selected

when ever we want to write Command / Data to LCD we have to first select that particular register.

- R/$\bar{W}$ (Read / Write signal)

we can use this to Perform read and write operation

1. when R/$\bar{W}$ = 0 write operation is performed.

which means we can write Command / Data on to Command / Data Register.

2. when R/$\bar{W}$ = 1 – Read a Data / Signal from LCD.

- En (Enable) - when ever we want to latch a data which means, 1st we will place a Data / Command on Data bus i.e (D0 to D7) after that we need to send a Enable signal.

En = 1 – 0 pulse (make the signal high and after some delay or immediately make it back to low this will do the write operation.

So to write a program for LCD interfacing first we have to select Command register / Data register which is done by using Rs control signal

2. place Data on Data bus D0 to D7
3. Select the operation to be performed i.e R/$\bar{W}$
4. Latch the Data / Command by sending high to low pulse on Enable signal.

* Commands of LCD:-

Various no of Commands are available for LCD Such as.

1 – clear display screen.
2 – Return home
4 – Decrement Cursor (Shift left)
6 – inc Cursor (Shift right)
5 – Shift display right and so on.

But generally 3 Commands are used in almost all the programs they are.

1. 38h → used to select LCD of 2rows of 5x7 matrix Display

which means to display each character on LCD Dot matrix pattern 5x7 is used.

2. 0Eh → Used to display ON, Cursor ON.

3. 01h → clear display.

- Before displaying any data on LCD we can send this Command to clear the precious display data on LCD
- generally the sequence of the Command written is same as mention above (1, 2, 3)

Next important Commands for LCD is related to address of character. As we are using 16x2 LCD we will have 16 characters that will be displayed on one row

∴ each character will have its own address
the add of 1st row starts from 80h, 2nd row = C0h.

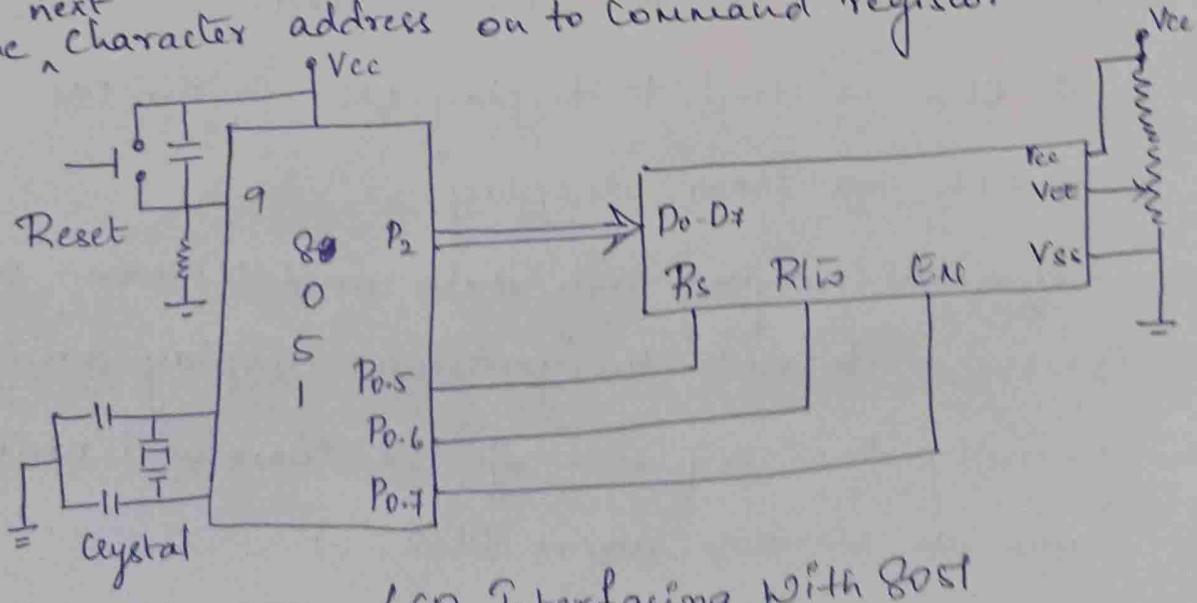
80h	81h	82h		. . .	
C0h	C1h	C2h	.	. . .	

- therefore to display characters at one particular position first we have to write address of that particular position in Command register

& GOD → I wanna display starting from character address 82h

82h	83h	84h
G	O	D

1. first write 82h in Command register
2. Send ASCII value of G and write it on to Data reg.
3. After writing G at 82h automatically cursor position will be incremented by 1 & cursor will point to next position. hence it is not necessary to write the next character address on to Command register.



LCD Interfacing with 8051

flow chart:- We will write a program to display a string on LCD using assembly level language.

- We have 8051, Reset CRT to reset the device.
- Crystal Oscillator which is connected externally
- Interfacing part of 8051 and LCD is shown
  - here D0 to D7 is connected to Port 2

  Any ports can be used to do the Interfacing.
- Rs (Register select) which is connected to P0.5
- R/W (Read/Write) is connected to P0.6
- En (Enable) is connected to P0.7
- we have connected a pot to alter the brightness of the LED.

1. Write a ALP to display a string on the first row of LCD.

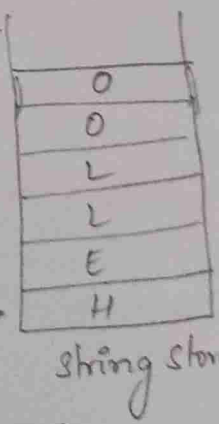
   * To write a program we use the concept of lookup table to store the data / string.

   the formate for this is

   String: DB 'HELLO', 0
   - 'HELLO' → Stores the 8bit value of each character in memory location
   - 0 → end of the string

* once the ASCII value of 1st char is read. on to Data Register, pointer is ↑ to the next location.

* there by every time the value is read from the memory location to which the pointer is pointing and then Compare that value will Zero. If not equal display the value if equal end the display program.



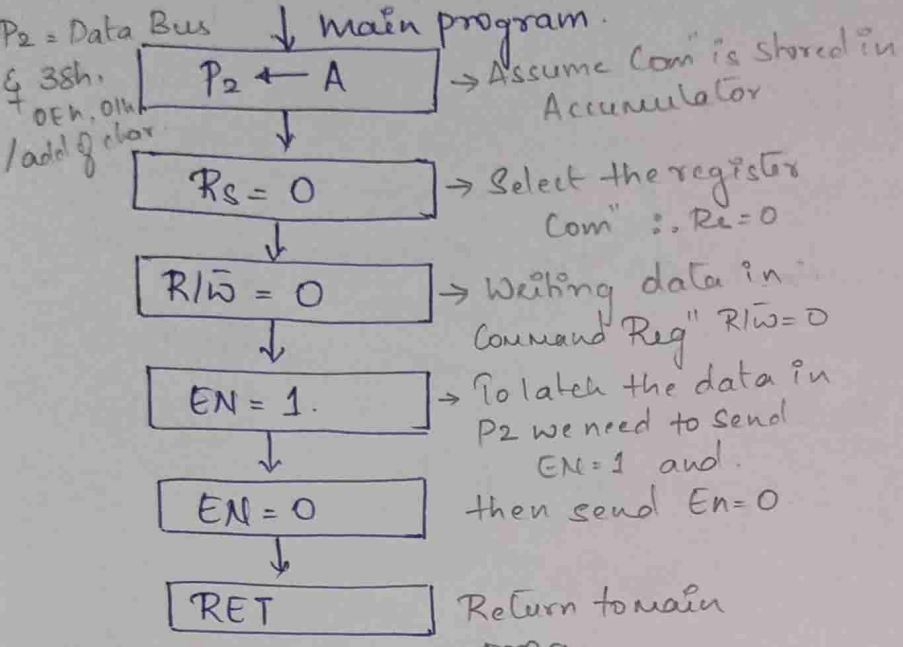
use pointer which will point to the memory location where 1st char is stored

String stored

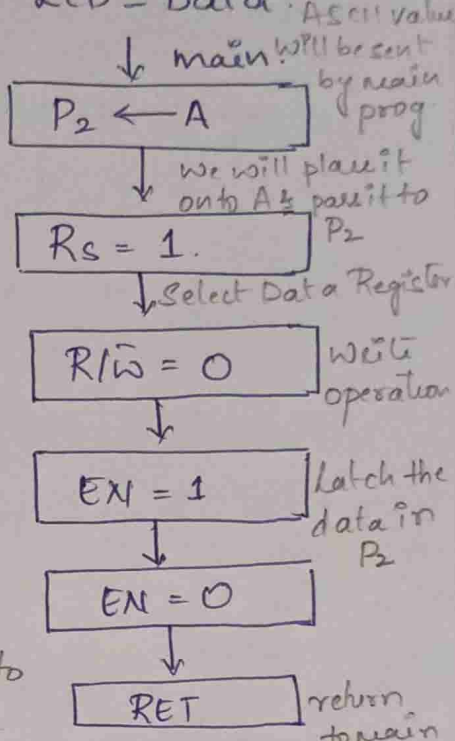
Flow chart :- we use 3 Subroutine 1. Delay 2. LCD_Command. 3. LCD_Data → Writes Data in Data register

Writes Command in Command Register

flow chart for LCD_Command.

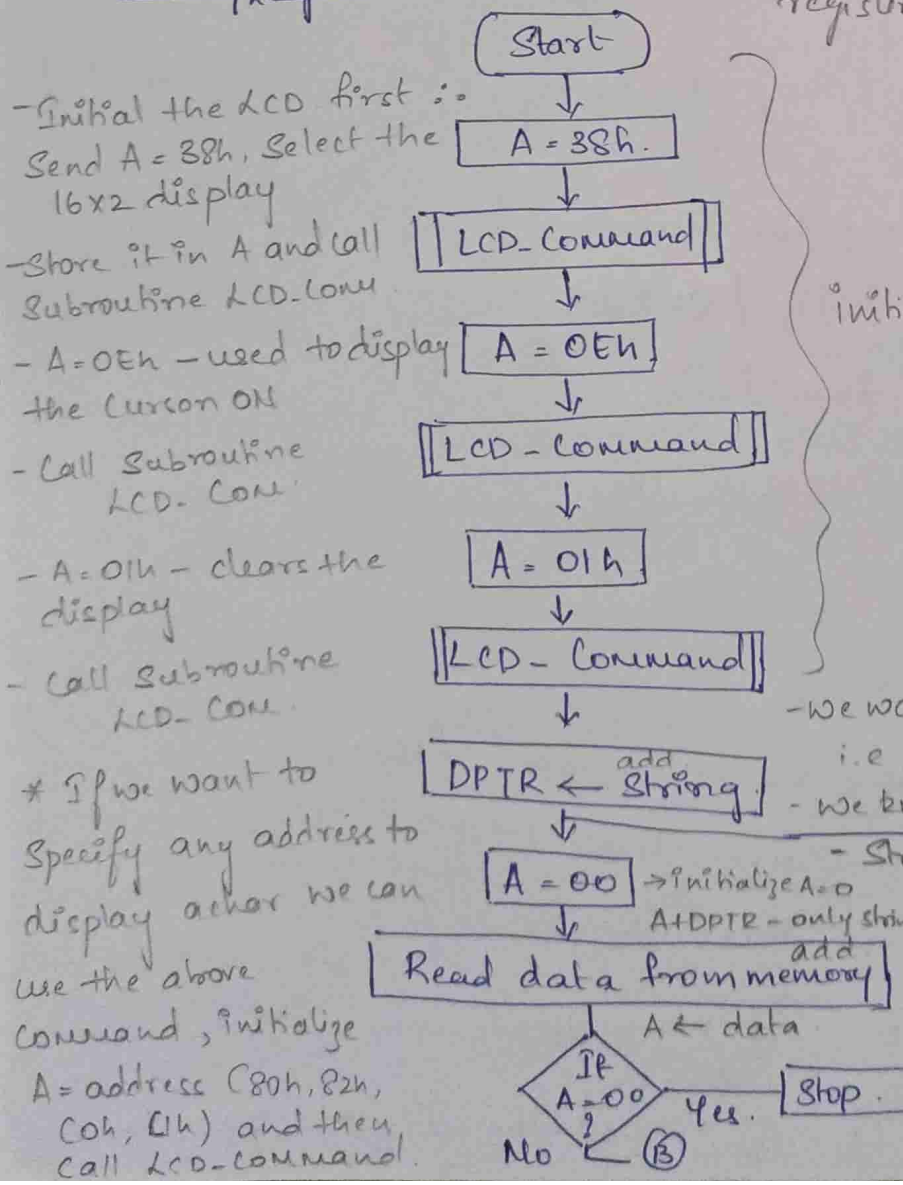


LCD_Data



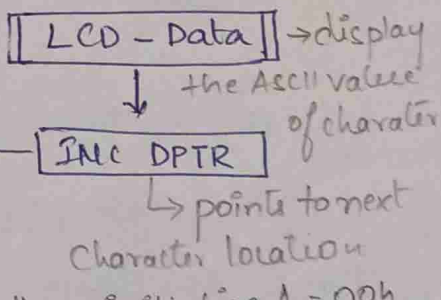
* Only difference is to Select the type of register

* main program :-



- Initial the LCD first :- Send A = 38h, Select the 16x2 display
- Store it in A and call Subroutine LCD_Com
- A = 0Eh - used to display the Curson ON
- Call Subroutine LCD_Com
- A = 01h - clears the display
- Call Subroutine LCD_Com

* If we want to Specify any address to display achar we can use the above Command, initialize A = address (80h, 82h, C0h, C1h) and then, call LCD_Command.



- we wana write data in data register i.e ASCII value of string
- we know we are storing the string
- String : DB 'HELLO', 0 - last inst in the program, Label: string ends with 0
- this inst will write the HELLO on to flash memory

To read data from flash
- MOVC A, @A+DPTR → point to string

A+DPTR - only string

① Write a ALP program to Send Command and data to LCD with a time delay.

```
; P2.0 to P2.7 are connected to LCD data pins D0 to D7
; P0.5 is connected to Rs pin
; P0.6 is connected to R/W pin
; P0.7 is connected to E pin.
Org 00h.
MOV A, #38h → init. LCD 2 lines, Selects 16X2 display.
ACall LCD_Command → Calls Subroutine to write Command.
ACall Delay → Apply time delay.
MOV A, #0Eh → LCD turns ON and Cursor ON.
ACall LCD_Command → Calls Subroutine.
ACall Delay → Apply time delay.
MOV A, #01h → Clears the previous display.
ACall LCD_Command → Calls Subroutine.
ACall Delay → Apply time delay.
MOV A, #84h → Cursor in Line 1. position 4
ACall LCD_Command → Calls Subroutine.
ACall Delay → Apply time Delay.
MOV A, #"O" → display letter 'O'
ACall LCD_Data → Call Subroutine to write Data.
ACall Delay → Apply time delay.
MOV A, #'N' → display letter 'N'
ACall LCD_Data → Call Subroutine.
ACall Delay → Apply Delay.
Again: Sjmp Again → Stay here.
       ↳ Subroutine to send Command to LCD.
LCD_Command: MOV P2, A → Content of A → Port 2.
             CLR P0.5 → Rs=0 ; Command Register selected
             CLR P0.6 → R/W=0 ; to write
             SETB P0.7 → E=1 ; latch the Data in Port 2
```

ACall Delay → Apply time delay.
CLR P0.7 → E=0
RET → Return to main program.

LCD-Data: MOV P_2, A → Content of A to Port 2.
SETB P0.5 → RS=1 ; Select Data Register.
CLR P0.6 → R/$\overline{W}$=0 ; Write Data.
SETB P0.7 → E=1 ; latch the Data in Port2
ACall Delay → Apply time delay.
CLR P0.7 → E=0
RET → Return to main program.

Delay: MOV R_3, #0FFh → Max value 255 → R_3.
L2: MOV R_4, #0FFh → 255 → R_4
Here: DJNZ R_4, Here → Stay till R_4=0
DJNZ R_3, L2 → Stay till R_3=0
RET → Return to main program.
End.

2. Write a Program to display string "HELLO WORLD" on 16X2 LCD using 8051.

```
Org 00h
MOV A, #38h
ACall LCD-Command.
MOV A, #0Eh
ACall LCD-Command.
MOV A, #01h.
ACall LCD-Command.
```
} initialization of LCD.

```
MOV A, #82h → Cursor in Line 1, position 2.
ACall LCD-Command.
MOV DPTR, #String → Point DPTR to string look up table. 1 char add" will be stored in DPTR
Back: MOV A, #00h → Assign DPTR with the content of string where Data that has to be displayed is stored.
MOVC A, @A+DPTR
```
00+ add" of 1 char

↑ Copy the data from memory location to A

```
        JZ exist → Jump if A=0 to exist
        ACall LCD_Data → ASCII value will written on A and the
                         data will be displayed on LCD.
        INC DPTR → next character.
        SJMP back → jump to back
exist : Sjmp exist. → Stay here.

LCD_Command : ACall delay.
        MOV P2, A
        CLR P0.5
        CLR P0.6
        SETB P0.7
        CLR P0.7
        RET

LCD_Data : ACall delay.
        MOV P2, A
        SETB P0.5
        CLR P0.6
        SETB P0.7
        CLR P0.7
        RET

Delay : MOV R3, #0FFh
   L2 : MOV R4, #0FFh.
Here : DJNZ R4, Here.
        DJNZ R3, L2
        RET

String : DB 'HELLO WORLD', 0
End.
```

* Interface ADC and DAC with 8051 micro controller :-

1. ADC devices :- Devices which are used to convert analog signal to digital signal.

ADC devices are most widely used for data acquisition.

- In physical world everything is analog & temp, velocity, pressure etc need to be converted to digital number so that the micro controller can read and process the info provided.

- ADC has n bit resolution, where n = 8, 10, 12, 16, 24 bits. High resolution ADC will have smaller step size.

Step size = smallest change that can be detected by an ADC.

n	No of step	Step size.
8	256	5/256 = 19.53mV
10	1024	5/1024 = 4.88V
12	4096	1.2 mV
16	65536	0.076 mV

5 = Vcc.

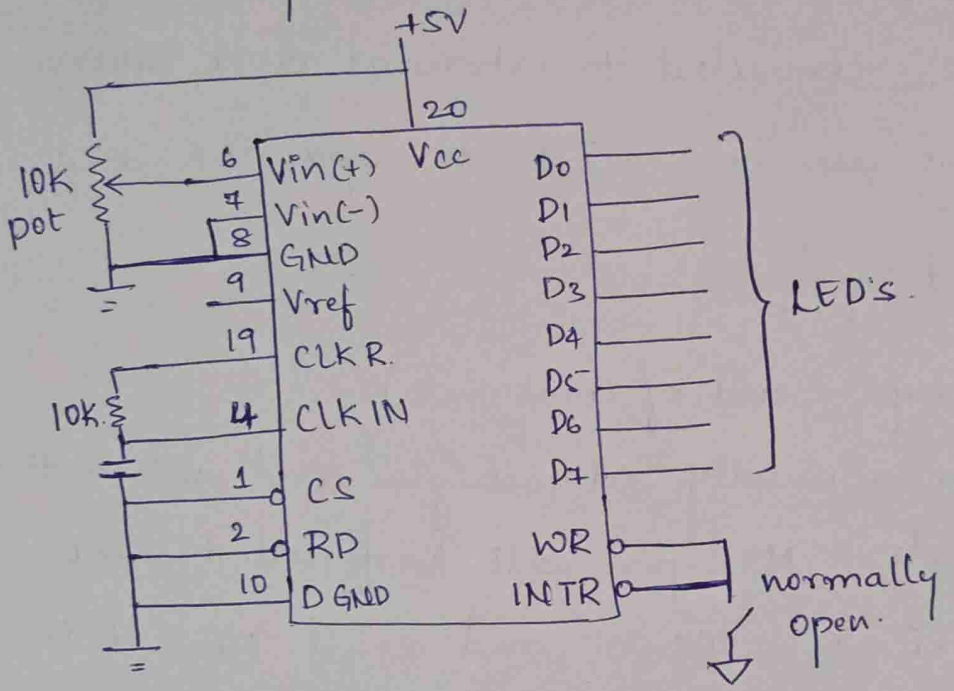
- Apart from resolution, Convecion time is another major factor in ADC.

Conversion time - time taken by ADC to convert analog i/p to a digital number.

- ADC chips are of two kinds.
  1. Parallel 2. Serial. → only one pin for data out
     ↳ 8 or more dedicated pins to bring out binary data.

1. Parallel ADC (0804) chip

- It is a 8bit parallel ADC in the family of ADC0800 Series from National Semiconductors.
- It works on +5 Volts and has a resolution of 8bits.
- Conversion time depends on the clocking signal applied at the clk in pin. (It cannot be faster than 110μs).



ADC 0804

1. CS → Chip Select
   this pin has to be set to low to activate the ADC chip.
2. RD → Read data.
   - ADC converts the analog i/p to its binary equivalent and stores the data in internal Register.
   - RD is used to get the Converted Data out of ADC when CS=0, if a high to low pulse is applied at RD, then digital o/p shows up at D0 - D7 datapins.
   - It is also refered as O/p enable (OE).

3. WR (Write / Start Conversion).

- If CS=0 & WR = low to high pulse.

then ADC starts to convert analog i/p value Vin to a 8bit digital number.

- time taken to convert depends on CLK IN & CLK R Values.

4. Clk In & CLK R

- clock pins connected to external clock source
- these pins are connected to capacitor and resistor

$$\therefore f = \frac{1}{1.1\,RC}$$

5. INTR (interrupt / end of conversion):-

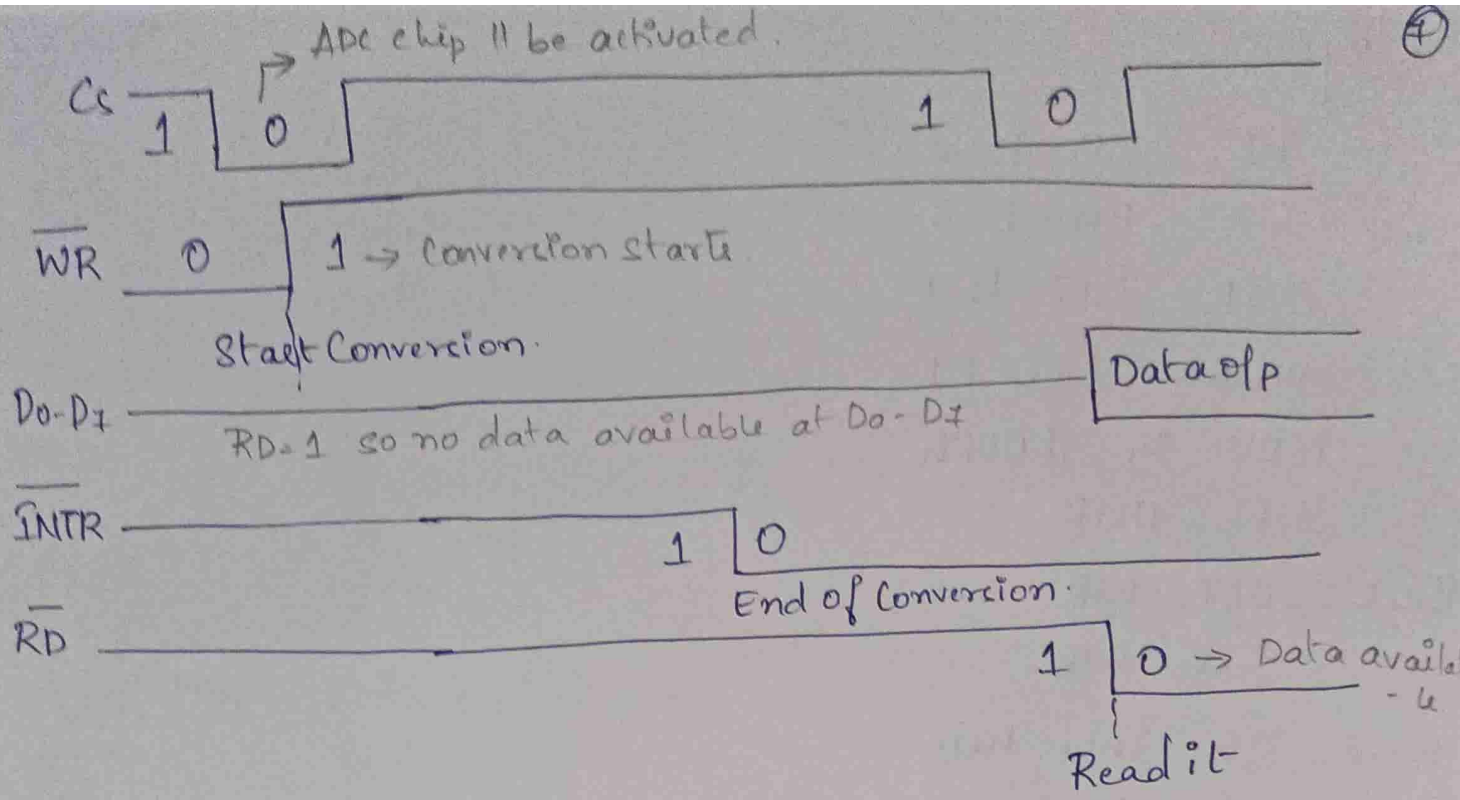
- This pin is usually set to high and when the conversion ends INTR pin will be forced to zero.
- once INTR=0, CS=0 and send high to low pulse to RD to get the data out of ADC chip.

6. Vin (+) & Vin(-)

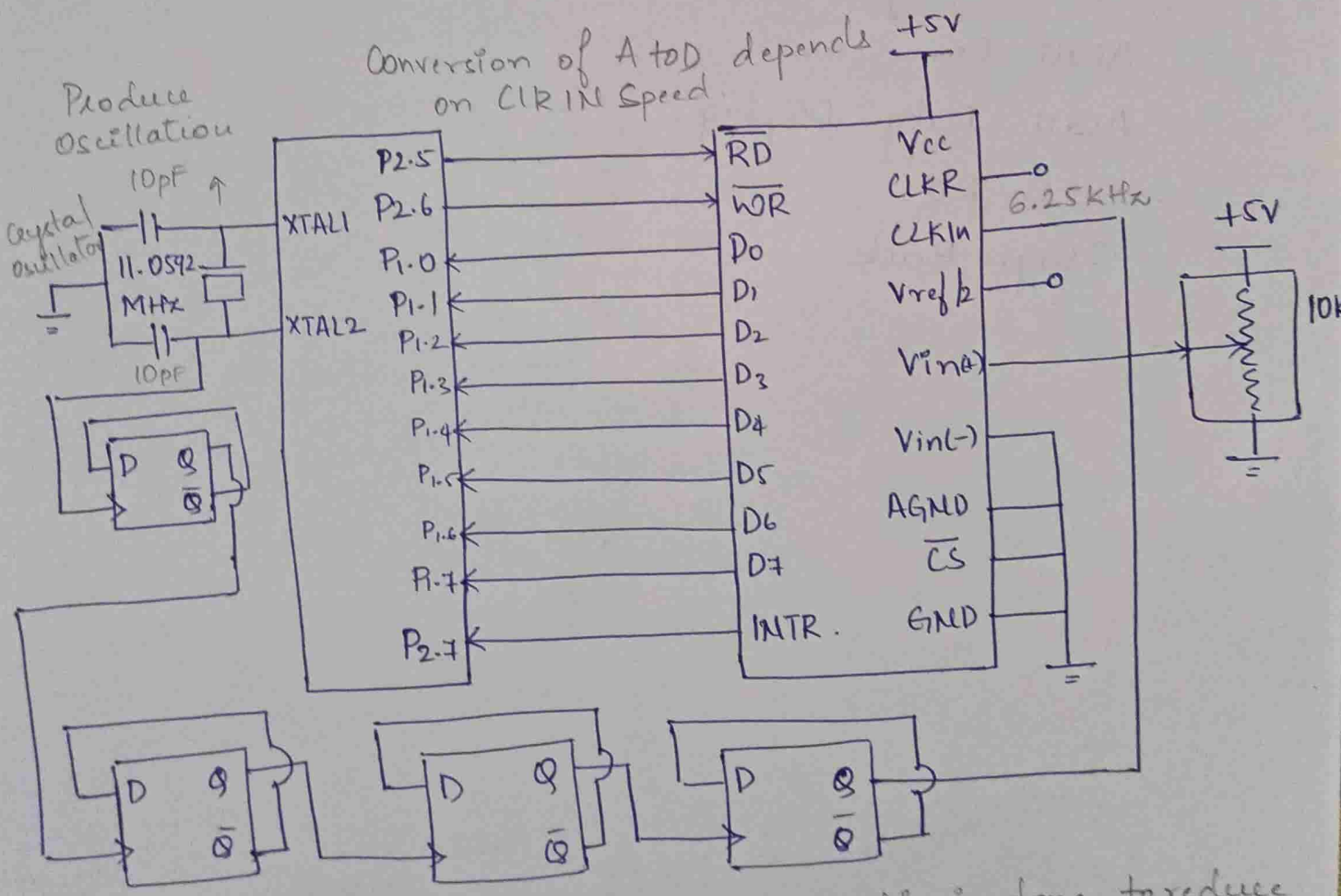
two different analog i/p

Vin = Vin(+) + Vin(-)

Vin(-) is connected to ground & Vin(+) is used as analog i/p to convert to digital.



* Connection to ADC0804 with clock from external clock



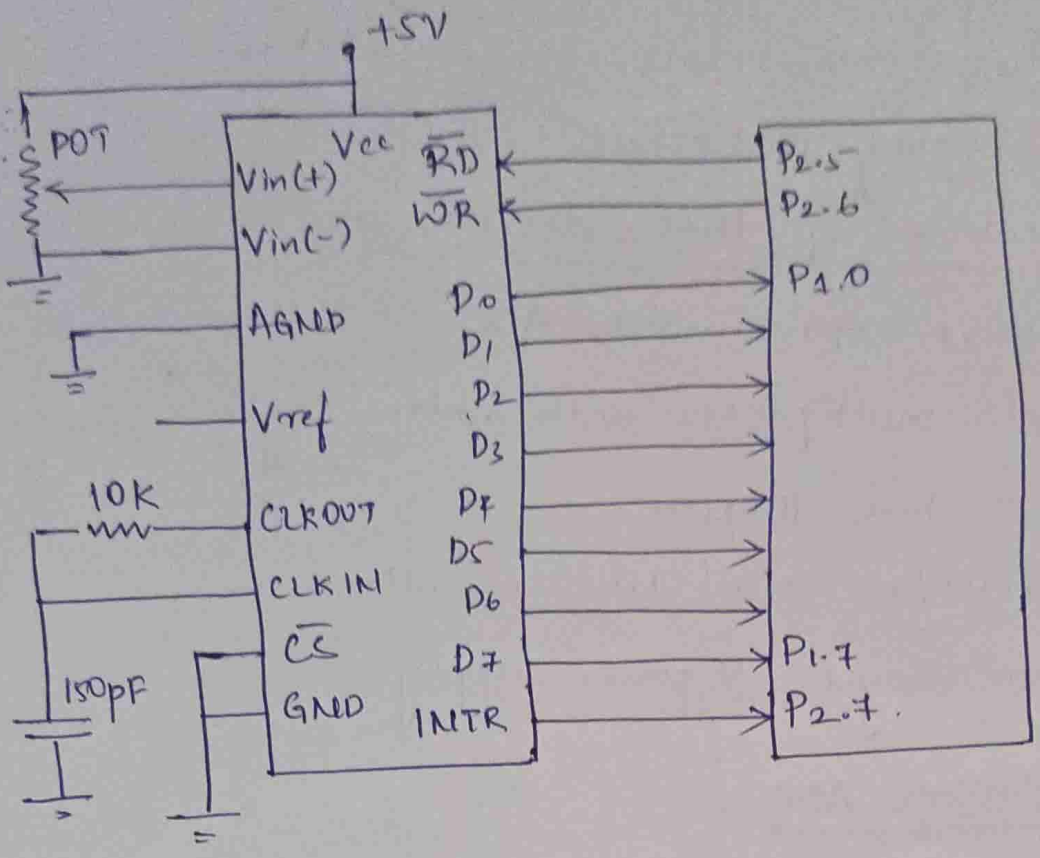
this is done to reduce the speed of the External clock so that we get the correct speed of the clock.

4 to 5 D flip flops are used. — to obtain the correct frequency for the ADC chip.

D flip flop divides the frequency by 2.

Program :-

```
        RD      BIT  P2.5
        WR      BIT  P2.6
        INTR    BIT  P2.7
        MYData  EQU  P1.
        MOV  P1, #0FFh.
        Setb INTR.
Back:   CLR  WR
        Setb WR.
here:   JB   Intr, here.
        CLR  RD
        MOV  A, Mydata.
        ACall Convercion.
        ACall Data_Display.
        Set B RD
        SJmp Back.
```



AD Conversion program :-

```
MOV P1,#0FFh ; P1 configured as i/p port.
Back: Clr P2.6      ; WR = 0
      Acall Delay   ; Call the Delay
      SetB P2.6     ; WR = 1. → indicate the start of
                      conversion.
Again: JB P2.7, Again ; P2.7 = 1 wait for the conversion
                        to end.
      CLR P2.5      ; RD = 0 - enable the read operation.
      MOV A, P1     ; Read the data to port 1.
      SetB P2.5     ; RD = 1 - disables the reading
                      operation.
      Sjmp back     ; go for the next conversion cycle.
```

* ADC 0808 / 0809 family :-

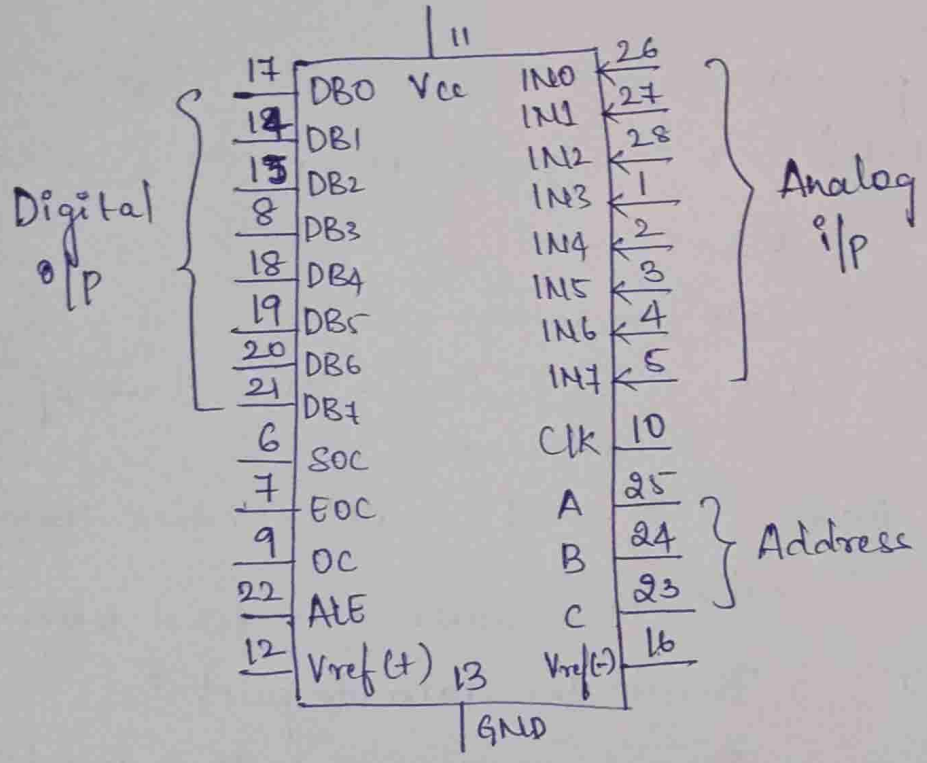
- then a monolithic CMOS devices with 8-channel multiplexer.

- they are designed to operate from a common microcontroller control bus, with tristate o/p latch driving data bus.

the main features of these device are.

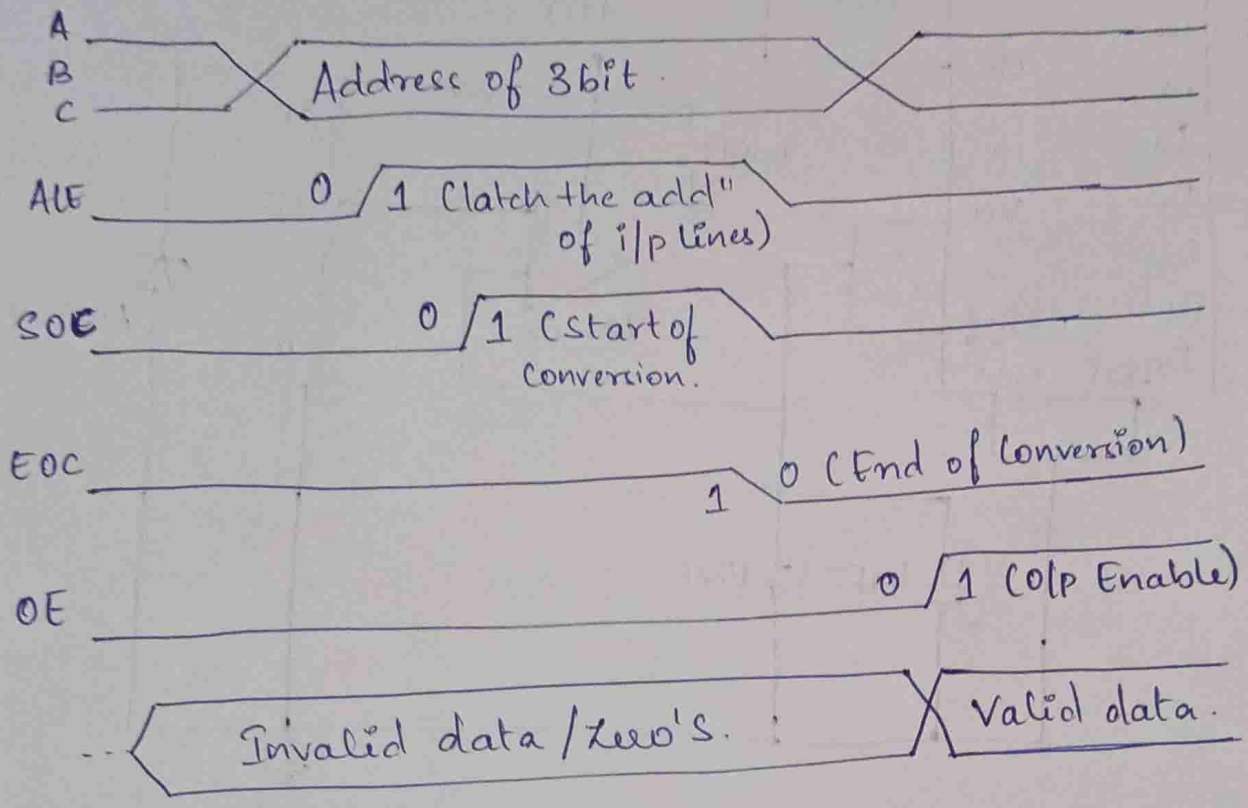
- 8bit Successive approximation ADC.
- 8-channel multiplexer with address logic.
- Conversion time 100μsec.
- Easy to interface to all microcontrollers.
- Operates on single 5V power supply.

Pin diagram of 0808 ADC.



- It has 8 i/p channels, to select desired i/p channel it is necessary to send 3bit address on A B C i/p pins.
- the Analog i/p is sent through Desired channel and the address of the i/p line is latched by sending ALE signal – low to high.

- To indicate start of conversion SOC must be switched from ~~high to~~ low. to high.
- Once the conversion is completed EOC [End of Conversion] must be forced from high to low.
- the converted digital o/p can be read through databus by enabling the o/p control pin (OE-1) after EOC is activated.



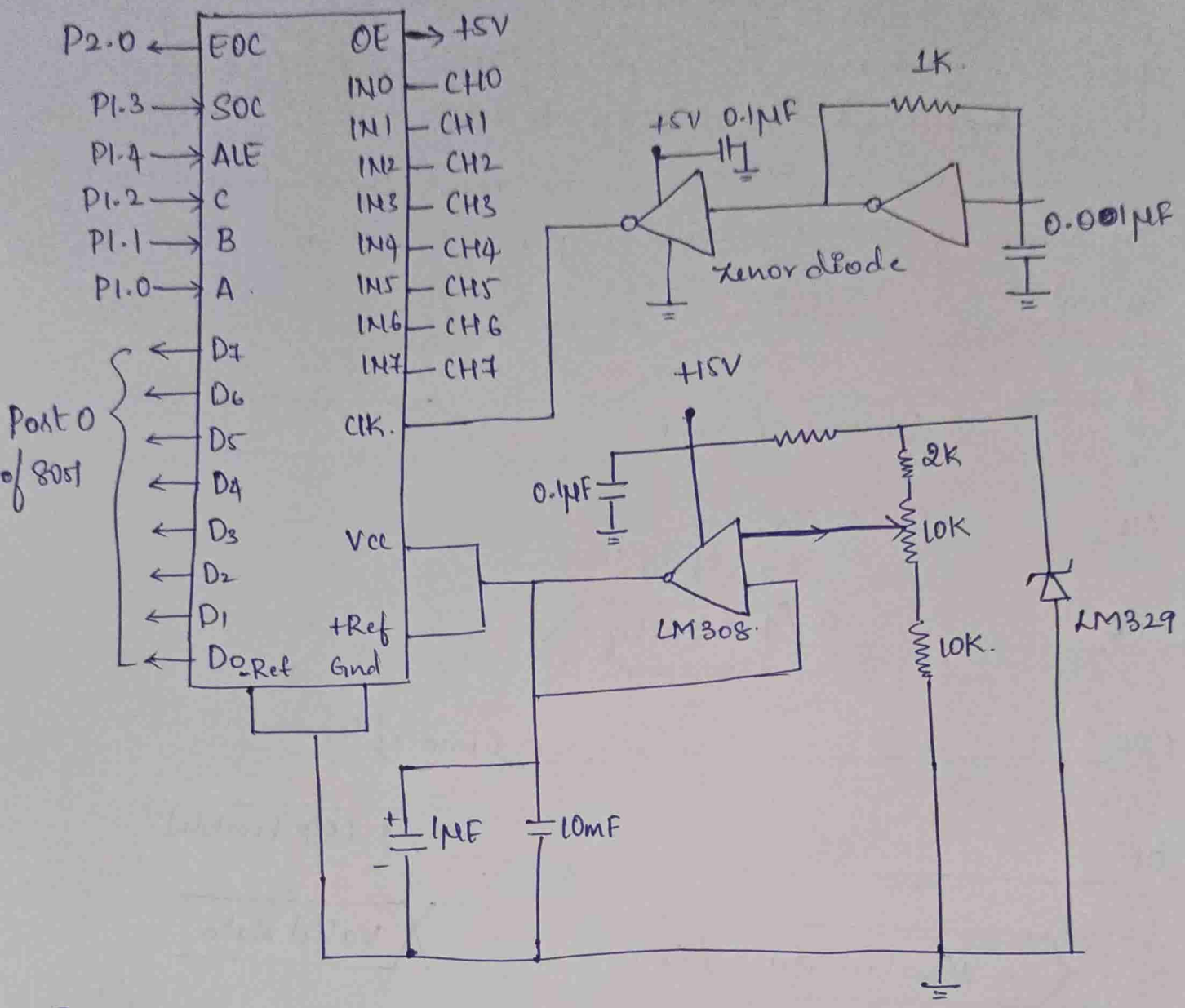
& Interface 8bit, 8channel ADC to 8051. Write a/p to convert ch0, ch3, ch7 and store result in external memory location starting from C000h. Repeat procedure by giving delay.

Step 1: Send 3bit address on ~~A~~C, B, A [P1.2, P1.1, P1.0]

Step 2: provide delay ALE - ~~low~~ high to low.

Step 3: SOC - high to low start conversion.

Step 4: EOC - activate
Send converted data.



Program :-

CLR P1.3 → Make SOC low.

CLR P1.4 → Make ALE low.

MOV P0, #0FFh → P0 as i/p port.

MOV P2, #0FFh → P2 as i/p port.

Back: MOV DPTR, #0C000h → memory pointer.

MOV A, #00h → Set address for ch0

ACall R_ADC → Call ADC Subroutine.

MOVX @DPTR, A → Converted data to DPTR location

INC DPTR → Increment DPTR.

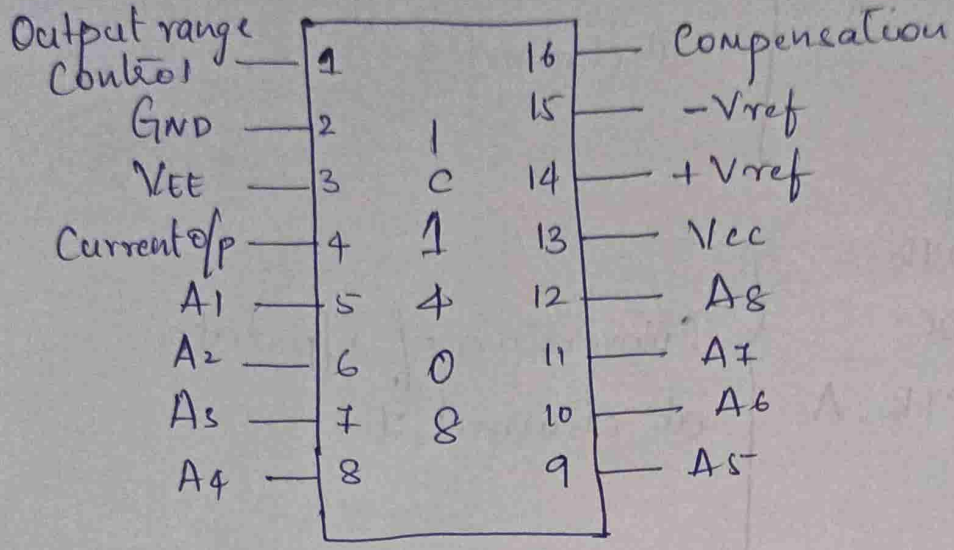
```
ACall Delay. → Call Delay Subroutine.
MOV A, #03h
ACall R_ADC
MOVX @DPTR, A      } Conversion of i/p data
INC DPTR.            at channel 3.
ACall Delay
MOV A, #07h.
ACall R_ADC
MOVX @DPTR, A      } Conversion of i/p data
INC DPTR.            at channel 7.
ACall Delay.
Sjmp back → Repeat.

R_ADC: MOV P1, A → get channel no and set its add"
       SetB P1.4 → ALE = 1 (latch the address)
       NOP       → Pulse delay.
       CLR P1.4  → ALE = 0
       SetB P1.3 → SOC = 1 (start of conversion)
       NOP       → Pulse delay.
       CLR P1.3  → SOC = 0
Wait : JB P2.0, Wait → wait till EOC = 0.
~~Wait1: JNB P2.0, Wait~~
       MOV A, P0. → get digital data onto A
       RET. → Return to main program.
```

* Interfacing DAC to 8051

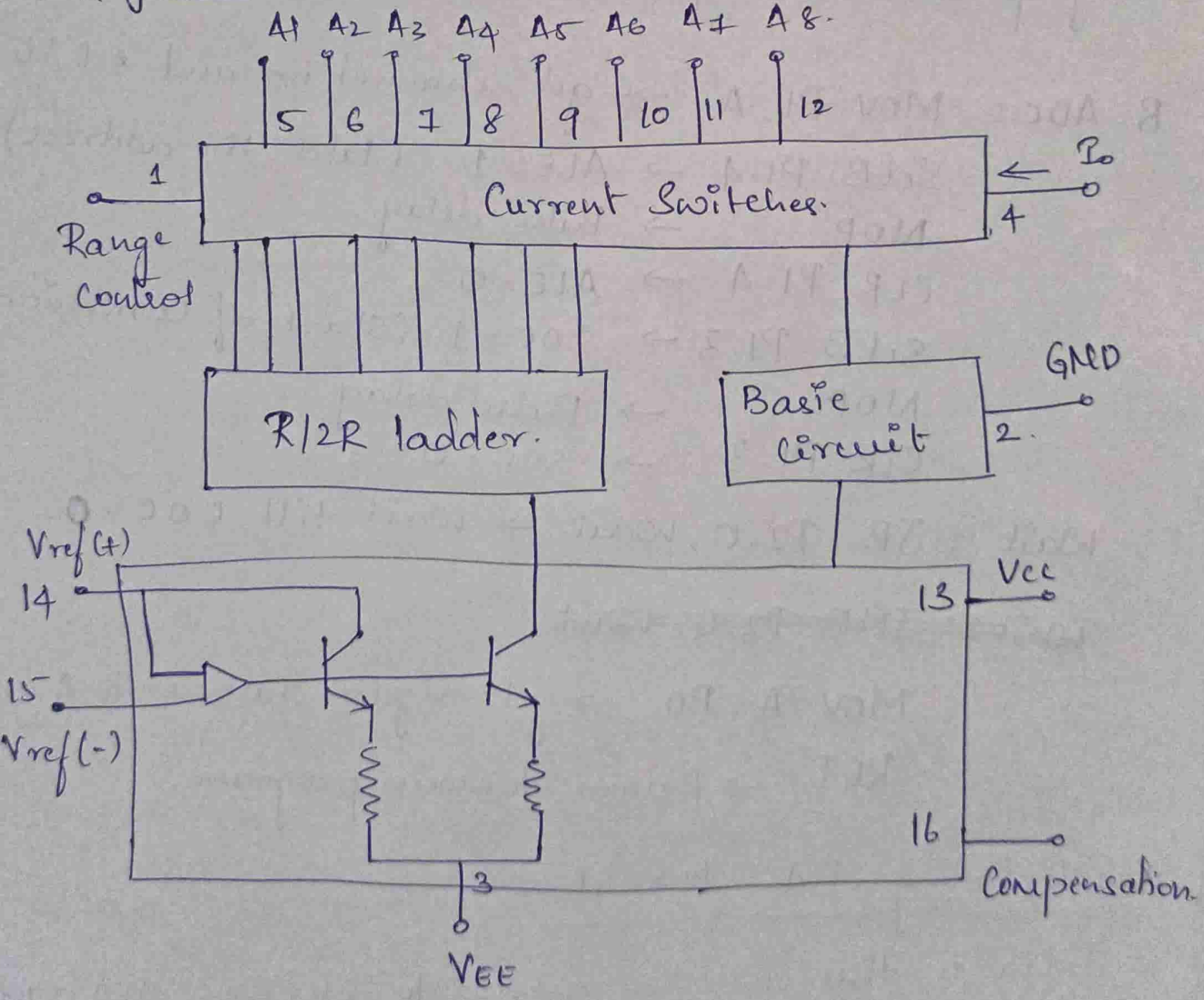
Under this we are going to interface DAC 1408, 8bit R/2R ladder type DAC with 8051.

- It is compatible with TTL and CMOS logic.

Pin diagram:-



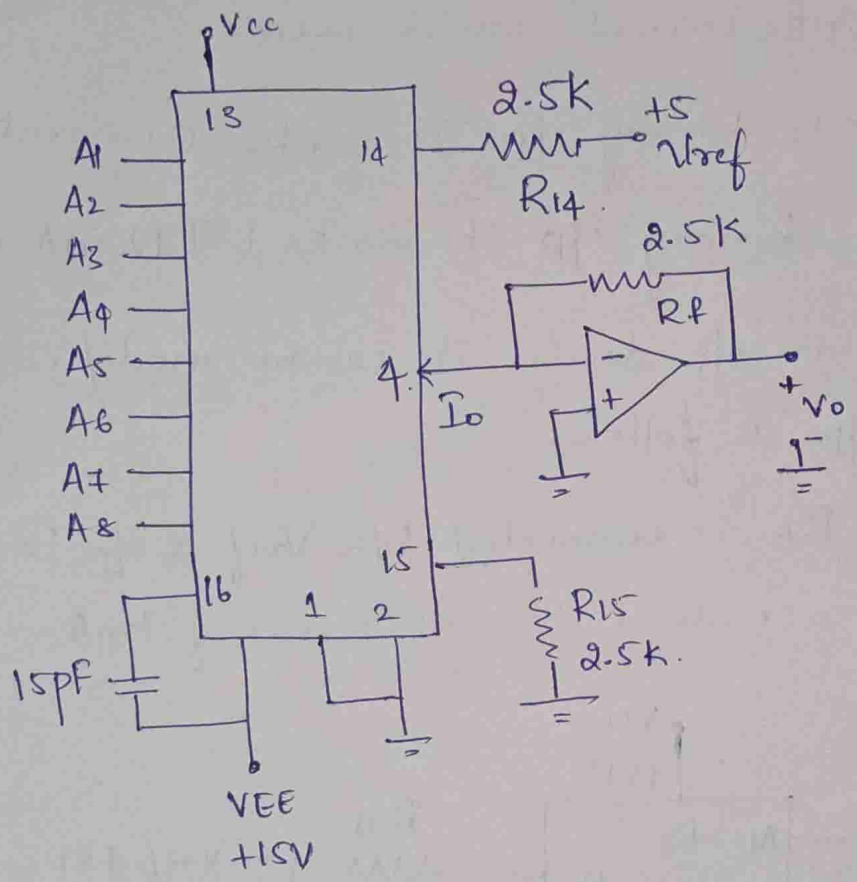
Block diagram:-



- 1408 consist of reference current amplifier.
R/2R ladder, 8 high speed current switch

- It has 8 i/p data lines A1 (MSB) to A8 (LSB) which control the position of current switches.
- It requires 2mA reference current from full scale i/p two power supply $V_{CC} = +5V$, $V_{EE} = -15V$

Atypical × Circuit × for IC 1408.



- Vref and R_{14} determines the total reference current source. and $R_{15} = R_{14}$ to match the i/p impedence of the reference current amplifiers.

∴ o/p current I_o is given by.

$$I_o = \frac{V_{ref}}{R_{14}}\left[\frac{A_1}{2} + \frac{A_2}{4} + \frac{A_3}{8} + \frac{A_4}{16} + \frac{A_5}{32} + \frac{A_6}{64} + \frac{A_7}{128} + \frac{A_8}{256}\right]$$

I/p A_1 through A_8 can be either 0 or 1. ∴ full scale current

$$I_o = 5/2.5K\left[1/2 + 1/4 + 1/8 + 1/16 + 1/32 + 1/64 + 1/128 + 1/256\right]$$

$= \frac{2mA \times 255}{256} = 1.992\,mA.$

the o/p voltage for full scale i/p is given by.

$V_o = 1.992 \times 2.5k = 4.98V.$

- Arrow on pin 4 is inwards. this means.

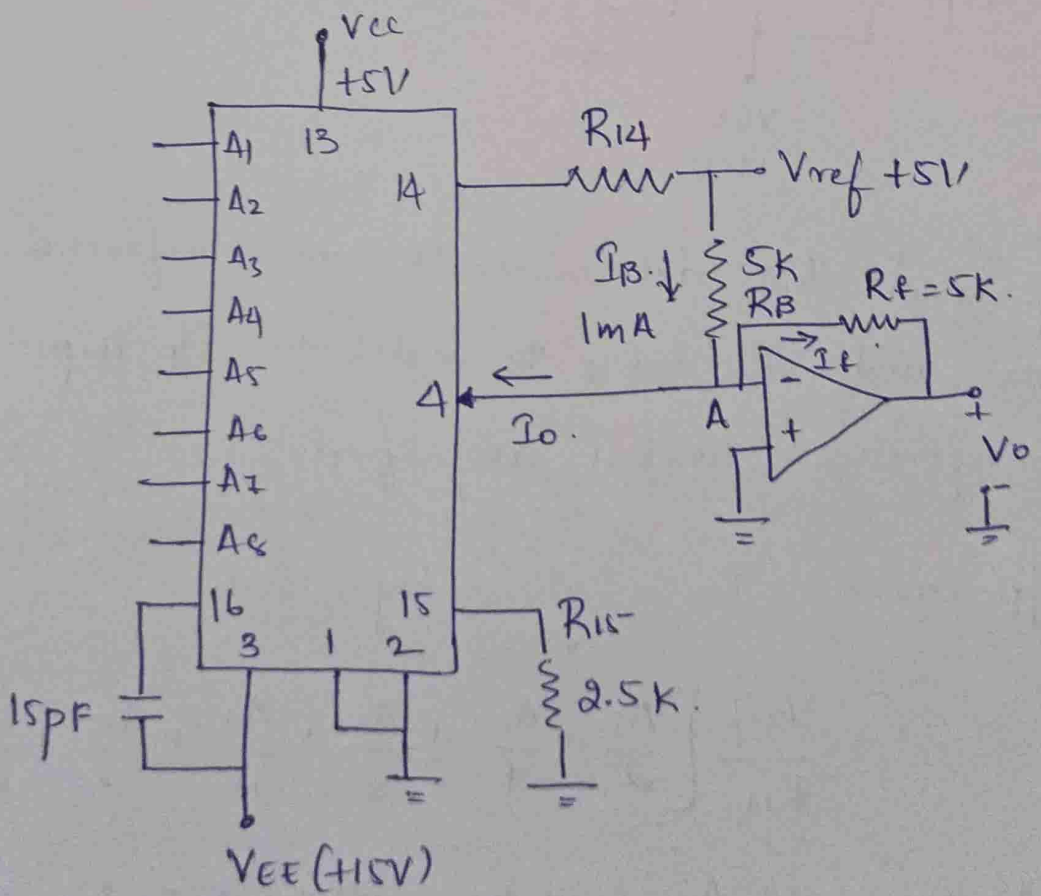
IC 1408 sinks current. which means;

At $(0000\ 0000)_2$ binary i/p it sinks 0 current

At $(1111\ 1111)_2$ binary i/p it sinks 1.992mA. current.

- It is a unipolar o/p device it can be modified to get a bipolar o/p as follows.

A resistor R_B is connected b/w Vref & o/p terminal. which gives a Constant Current Source of 1mA.



Operation can be observed for three Condition.

1. Binary i/p (00H)

When i/p = 00h o/p current I_o @ pin 4 = 0.

Due to current flow in R_B, the same flows through R_f giving $V_o = -5V$.

$V = V_{ref}\left[\frac{D}{2} + \frac{D}{4} + \quad\right]$

2. for Binary i/p 80h. [1000 0000]

i/p = 80h., o/p current $I_o = 1mA$. By applying KCL at node A. we get.

$-I_B + I_o + I_f = 0$.

$I_B = 1mA$

$I_o = 1mA$

$-(1mA) + (1mA) + I_f = 0$

$I_f = 0$.

∴ o/p Voltage is Zero.

3. for Binary i/p FFh. [1111 1111]

o/p current $I_o = 2mA$, apply KCL @ node A.

$-I_B + I_o + I_f = 0$

$-1mA + 2mA + I_f = 0$

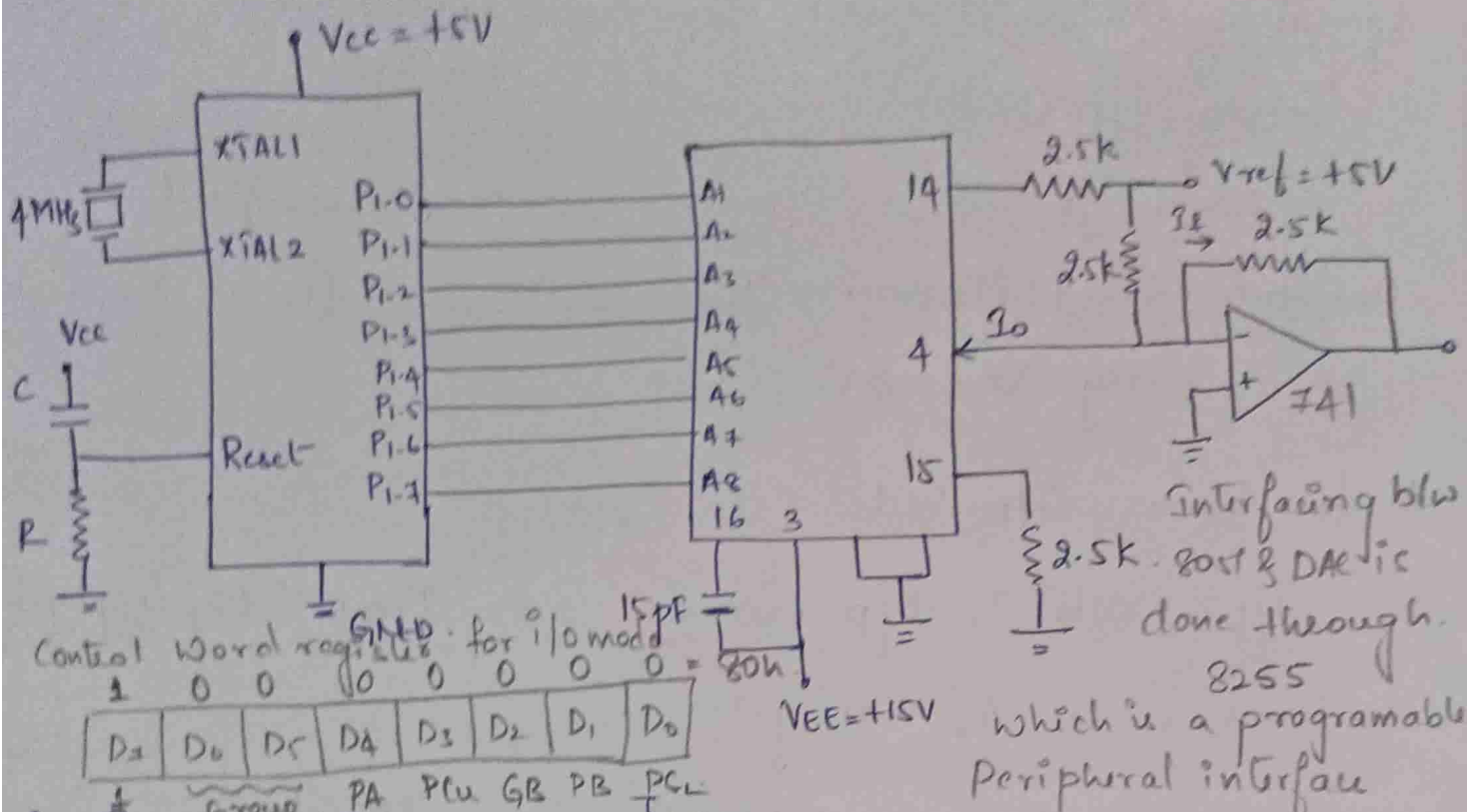
$I_f = -1mA$.

∴ o/p voltage is +5V hence o/p is Bipolar.

* Interfacing DAC 1408 / DAC 0808 with 8051

- microcontroller generate o/p which is in the digital for.

Ie 1408/0808 Converts digital data into eq" analog Current



Interfacing b/w 8051 & DAC is done through 8255 which is a programable Peripheral interface

Control word register for i/o mode

1	0	0	0	0	0	0	0 = 80h
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
↓	Group selection		PA	PCu	GB	PB	PCL
mode							

- D_7 = 1 for i/o mode
- D_6 & D_5 = 00 mode 1 = Simple i/p
  01 - mode 2
  1x - mode 3
- PA - 1 i/p , 0 - o/p
- Port C upper (PCu) } 1 - i/p
  Port C lower (PCL) } 0 = o/p
- Group B mode selector
  PB - 1 i/p , 0 = o/p
- PA = 0 we donot accept i/p we just sent data to convert

1. Square wave generation.

To generate Square wave we have to FFh & then 00h on port 1.

```
Program:  Org 00h
Repeat: MOV P1, #0FFh  ; Load all 1's in port 1
        Lcall Delay
        MOV P1, #00h   ; Load all 0's in port 1
        Lcall Delay
        Ljmp Repeat    ; Repeat
Delay:  MOV R0, #0FFh  ; Load delay count
Back:   DJNZ R0, Back  ; Decrement & check whether delay count is zero if not repeat the operation.
        RET
```

2. triangular wave :-

1 o/p data = 00h then increment to FFh

2. when reached FFh then decrement to 00h.

Program :-

```
Org 00h.
MOV R0, #00h.
back : MOV P1, R0  ; Send digital data to i/p of DAC
       INC R0      ; increment digital data
       CJNE P1, #0FFh, back ; check for peak value
next : MOV P1, R0  ; Send Peak value to P1
       DJNZ R0, next ; Decrement digital data
       Sjmp back.  ; jump to repeat the same  (R0 = 0)
       End.
```

* Sine Wave generation :-

- To generate sine wave we have to o/p digital equivalent values which represents a wave [Sine]
- Digital data 00H represents -2.5V , 80h represent 0V and FFh represents +2.5V
- digital equivalent for is calculated as shown.

  Sin0 = 0 Sin90 = 1.

∴ the range Sin0 to Sin90 is distributed over the digital range 80h to FFh (80+FF) = 128 decimal steps. (7F = 127)

∴ Digital equivalent for $\sin x = (128 + 128 \times \sin x)$ (127 + 127 × sin x)

where x is the angle in degrees.

Sin 0 = 128 + 128 x sin 0 = 128 = 80h (7F)

Sin 10 = 128 + 128 x sin 10 = 150 = 96h

Sin 90 = 128 + 128 x Sin 90 = 256 → 255 = FFh.

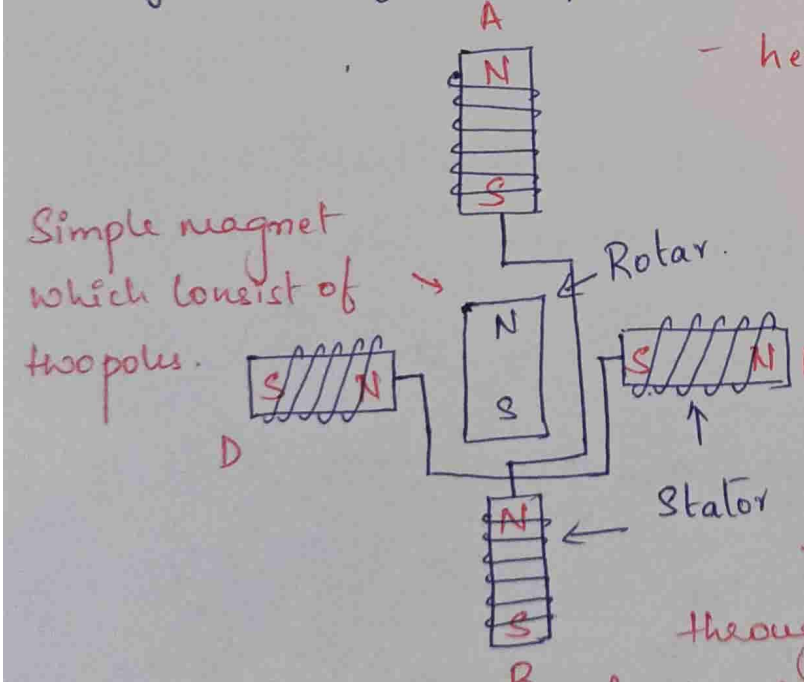
Sin 180 = 128 = 80h, Sin 270 = 0 = 0h Sin 360 = 128 = 80h.

Program :-

```
      CLR A
again: MOV DPTR, #Sine   ; Initialize the pointer to
       MOV R0, #24       ; Counter          lookup table.
Begin: CLR A
       MOVC A, @A+DPTR   ; data from lookup table to A
       MOV P1, A         ; Move it to port 1.
       INC DPTR          ; point to next lookup table entry
       DJNZ R0, Begin    ; Decrement counter & repeat.
       Simp again        ; Repeat entire process.
       127
Sine : DB, 128, 150, 192, 217, 237, 250, 255, 250,
       237, 217, 192, 150, 128, 94, 63, 37,
                           127
       17, 4, 0, 4, 17, 37, 63, 94, 128, 0
                                    127
End.
```

* Stepper motor interface :-

Stepper motor is a digital motor driven by digital signal. A typical 2 phase motor is as shown below.



Simple magnet which consist of two poles.

- here we have a Rotar surrounded by 4 stator.
- A stator is an electeo magnetic substance which is winded by coils.
- when current is applied through this coil then magnetic field of these magnets will be active
- A and B are connected by common wire

- C and D connected by common wire.
- When current is applied through A then magnetic field of Both A and B will be activated.
- When current is applied through B then the direct of current changes and hence polarity of magnets will be reversed.

1. Full step Sequence.

- We have 4 Stators A, B, C, D.
- In full step sequence we perform 4 steps to complete 1 Rotation which means in each step the Rotar rotates by 90° angle.

A	B	C	D	
1	0	1	0	QA
0	1	1	0	06
0	1	0	1	05
1	0	0	1	09

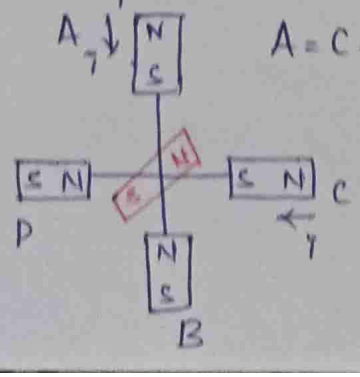
- Initially we assume current is flowing through A and C.

A = 1 B = 0, C = 1, D = 0

∴ the N pole of rotar will be equally attracted by the S pole of A and C.

the S pole of rotar will be equally attracted by the N pole of B and D.

∴ the position will be

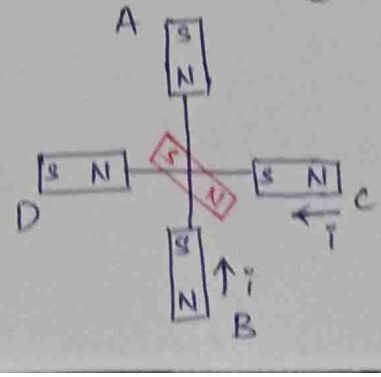


A = C = 1

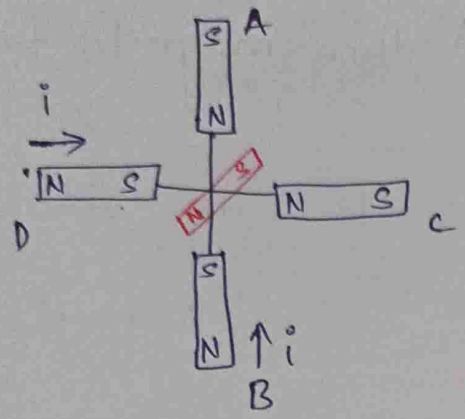
B = C = 1.

Current applied through B

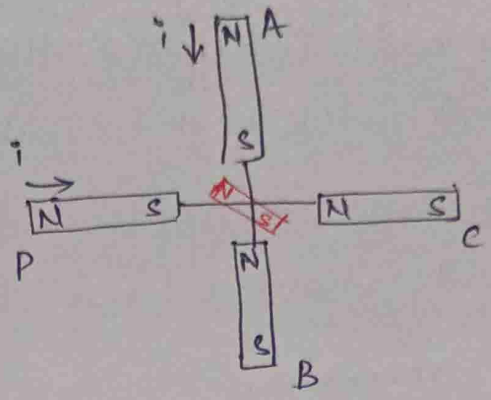
∴ polarities change



B = D = 1.



A = D = 1



∴ Rotar has Completed 1 Rotation

- Half Step Sequence then 8 Steps to Complete 1 Complete Rotation and Rotar Rotates with an angle of 45°

A	B	C	D	
1	0	1	0	0A → A and C = ON Supply Current
0	0	1	0	02 → A & B = inactive hence rotar will
0	1	1	0	06 be horizontally placed as it is
0	1	0	0	04 ★ C&D attracted only By C and D.
0	1	0	1	05
0	0	0	1	01 → AB · inactive
1	0	0	1	09
1	0	0	0	08 → C&D inactive.

- If we observe the rotar Rotates in clock wise direction in case we want it to rotate anticlock wise apply the current in reverse order.

Q1. Write an 8051 ALP program to control stepper motor

```
Org 00H
Ljmp 100H
Org 100H
MOV DPTR,#110H ; DPTR with memory location 100H
MOV A, #0AH
MOVX @DPTR, A
MOV P1, DPTR
```

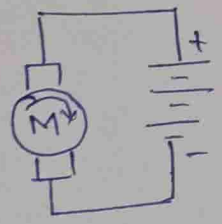
Solution:

```
MOV R0, # count  ; initialize the rotation count
Again: MOV DPTR, # Data ; point to the data in lookup table
Back:  MOV R1, #04 ; Counter to get data from look up table
       MOVX A, @DPTR ; get the data from code
       MOV P1, A ; Send the data to the port
       ACall Delay ; given delay to observe the slow rotation
       INC DPTR ; increment the pointer
       DJNZ R1, Back ; R1 ≠ 0, jump to back
       DJNZ R0, Again ; R0 ≠ 0 jump to Again
       Sjmp $ → looped jumping ; Sjmp here
Data: DB 0AH, 06H, 05H, 09H, 0; code sequence for clock wise rotation
```

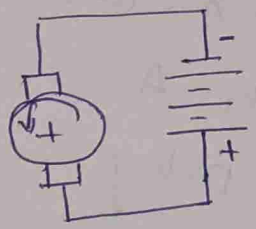
* DC motor interfacing:-

- DC motor → Direct current motor which rotates continuously.
- It has two terminals +ve & -ve.

- Connecting DC power supply to these terminals rotate motor in one direction an on reversing the polarity of the power supply reverses the direction of rotation.
- Speed is measured in rotation per minute (RPM).
- Speed increases with increase in the supply voltage.
- Speed of DC motor depends on the load
  - At no load speed is high.
  - As load ↑ speed ↓
  - Overloading will damage DC motor because of the excess heat produced by the load.

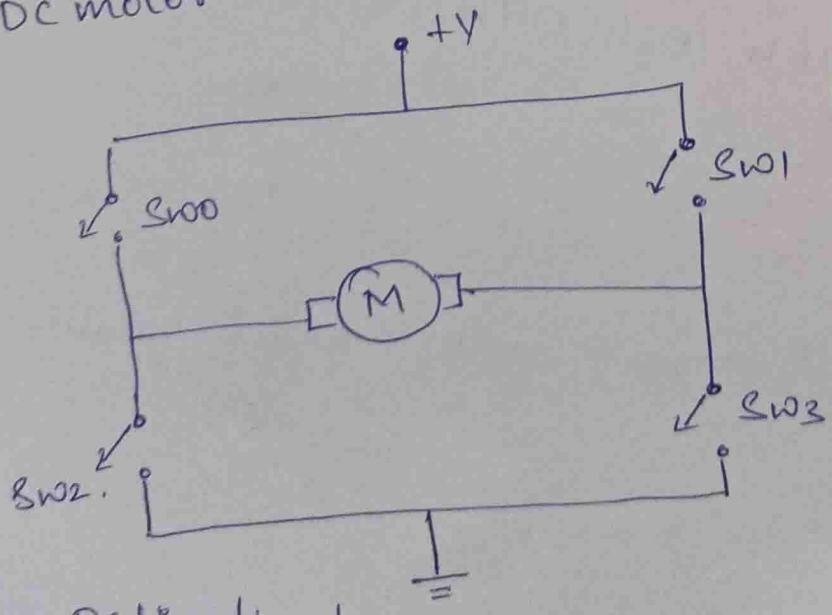


Clockwise rotation



anti clockwise.

By using switches for changing the power supply we can control the direction of rotation. of the DC motor.



Bidirectional

when all the switches are off motor doesnot run.

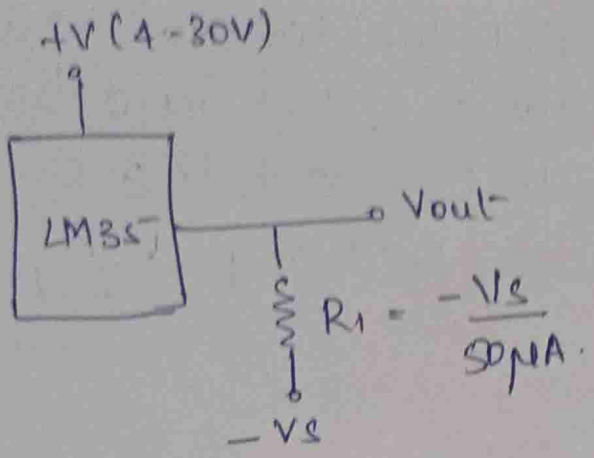
SW3	SW2	SW1	SW0	
ON	OFF	OFF	ON	clock
OFF	ON	ON	OFF	anti clock

Write an ALP to control the direction of the DC motor according to the status of bit P1.0. Assume the port pin P2.0, P2.1, P2.2 and P2.3 control the switches SW0, SW1, SW2, SW3 respectively. (16)

```
        Org 00H.
        CLR P2.0
        CLR P2.1
        CLR P2.2        } make all switches OFF
        CLR P2.3
        SetB P1.0
Check:  JNB P1.0, clockwise. ; jump if P1.0 ≠ 1
        CLR P2.0  ; SW0 = OFF
        SETB P2.1  ; SW1 = ON
        SETB P2.2  ; SW2 = ON
        CLR P2.3  ; SW3 = OFF
        Sjmp check.
clockwise: SetB P2.0 ; SW0 = ON
        CLR P2.1 ; SW1 = OFF
        CLR P2.2 ; SW2 = OFF
        SetB P2.3 ; SW3 = ON
        Sjmp check.
        End.
```

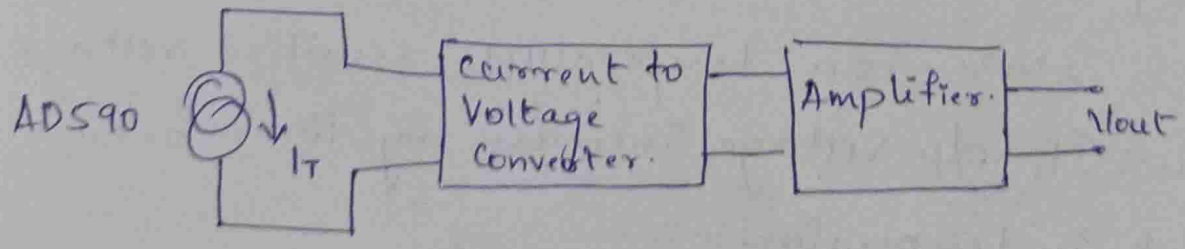
* Temperature Sensor [LM35]:-

LM35 is a temperature sensitive voltage source. Its o/p voltage increases by 10mV for each °C ↑ in temperature.



- o/p voltage is connected to a -ve reference voltage. this sensor will get a meaningful o/p for a temperature range of -55°C to 150°C
- o/p is adjusted to 0V for 0°C.
- o/p voltage from LM35 is applied to ADC to get digital equivalent of analog voltage.

* AD590 - temperature sensitive current source.
- It produce a current of 1μA/°K.
- this can be converted to voltage by using a current to voltage converter.
- Advantage is voltage drop in long connecting wire doesnot effect the o/p value.



the o/p of amplifier is applied to ADC to get digital equivalent of current temp.